Ingénierie des Systèmes d'Information

Chap. 7 : UML

Principes généraux et applications SI

Erwan TRANVOUEZ
erwan.tranvouez@univ-amu.fr

POLYTECH MARSEILLE
Aix*Marseille Université

MOD Global : ***Cas des cardinalités***

- Comment évaluer ces critères
  - **Par analyse historique** : on reprend les données réelles (données papiers, données déjà informatisées, analyse marché, …) et on calcule …
  - **Par approximation mathématique** : reconnaissance d'une loi de distribution de probabilités et extrapolation. (EX. loi de répartition rectangulaire appliquée à la distribution du nombre de participations).

Plan du cours

- **Session 1 :** Problématique de l'Ingénierie des SI
- **Session 2 :** Présentation générale de MERISE
- **Session 3 :** Modélisation des Traitements : DF & MCT
- **Session 4 :** Modélisation des Données : MCD
- **Session 5 :** Modélisation Organisationnelle des Traitements
- **Session 6 :** Modélisation Organisationnelle des Données
- **Session 7 :** Modèles Logiques : Traitements-Données
- **Session 7 bis :** Partiel
- **Session 8 :** UML 1
- **Session 9 :** UML 2
- **Session 10 :** UML 3

Objectifs de la session :

- Appréhender le formalisme UML
- Comprendre sa mise en œuvre dans le cadre d'un projet de développement de système informatique avec un paradigme Objet.

- Biblio centrée UML:
  - **Modélisation objet avec UML**, P-A Muller et N. GAERTNER, ed. Eyrolles.
  - **De Merise à UML**, N. Kettani, D. Mignet, P. Paré, C. Rosenthal-Sabroux, Ed. Eyrolles
  - **UML en Français**, http://uml.free.fr
  - **UML 2 par la pratique** : études de cas et exercices corrigés. P. Roques. Ed Eyrolles.
  - Diagramme UML réalisés avec Poseidon for UML – Community ed. – v 3.2..

Plan de la session

- **Présentation générale UML**
- **Diagrammes UML**

1. Présentation Générale d'UML

Définitions et enjeux

UML Origine

- Unified Modelling Language : **langage** de modélisation orienté objet
- Issu (principalement) de la synthèse de 3 méthodes influentes dans les années 90 : OMT (Object Modelling …), Booch et OOSE (Object Oriented Software Engineering). Début des travaux 1995.
- Devenu un standard dans la modélisation objet, promu par l'OMG (Object Management Group) qui a défini la norme CORBA par ex.
  - 1997 : UML 1.0

UML Objectif

- Finir avec l'hétérogénéité des méthodes, outils et langage de modélisation objet.
- Suivre une logique de norme avec dans l'idée que l'informatique n'est pas une industrie si différentes des autres…

  => Faciliter la communication entre concepteur / développeurs

  => Justifie (a posteriori ☺) la démarche rigoureuse du projet
- Indépendant de tout langage de programmation…
- … tout en laissant ouvert des solutions de « traduction »…

Qu’est ce qu’UML

- Un langage de modélisation assistant la production de logiciel à l’aide du paradigme objet.
- S’appuie pour cela sur des vues du système à construire
  - Vue statique/dynamique
  - Vue analyse/déploiement
  - …
- Et des diagrammes permettant de décrire de manière graphique ces vues.

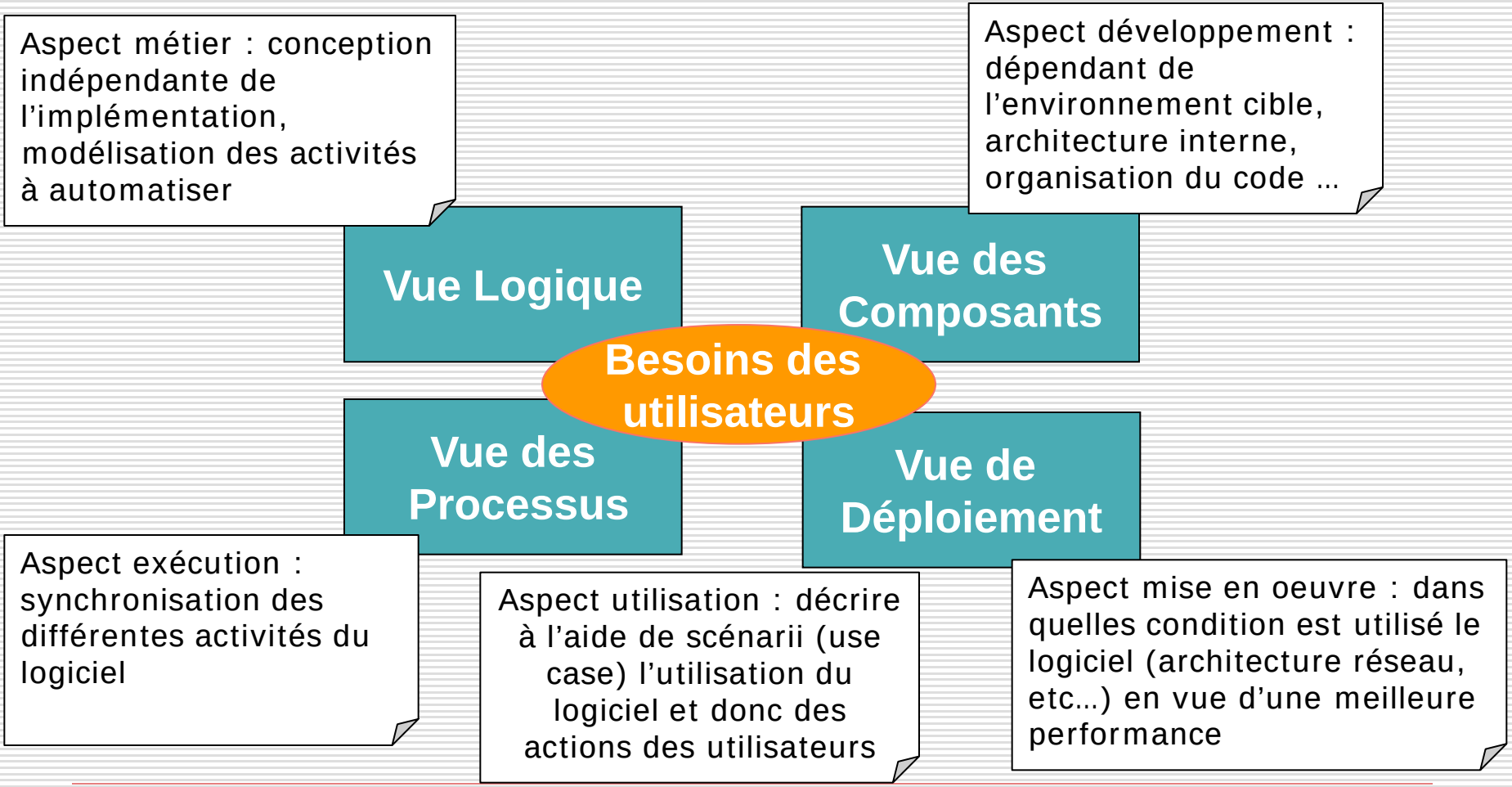
Ce que n’est pas UML

- Une méthode !
  - … Même si une méthode minimale peut consister en une séquence de diagramme UML.
- Mais le langage de modélisation se révélant relativement générique, d’autres champs de modélisation s’en sont emparés:
  - Paradigme agent/multi-agents: avec AUML utilisé par la FIPA (Foundation for Intellignent and Physical Agent) elle-même faisant partie d’IEEE.
  - Modélisation d’entreprise : avec EUML

Ce que propose UML

- Vue fonctionnelle du système
  - **diagrammes de cas d'utilisation**
- Vues statiques du système :
  - diagrammes d'objets
  - **diagrammes de classes**
  - diagrammes de composants
  - diagrammes de déploiement
- Vues dynamiques du système :
  - **diagrammes de collaboration**
  - **diagrammes de séquence**
  - diagrammes d'états-transitions
  - diagrammes d'activités

Démarche préconisée par UML



2. Vue Fonctionnelle

Use Case

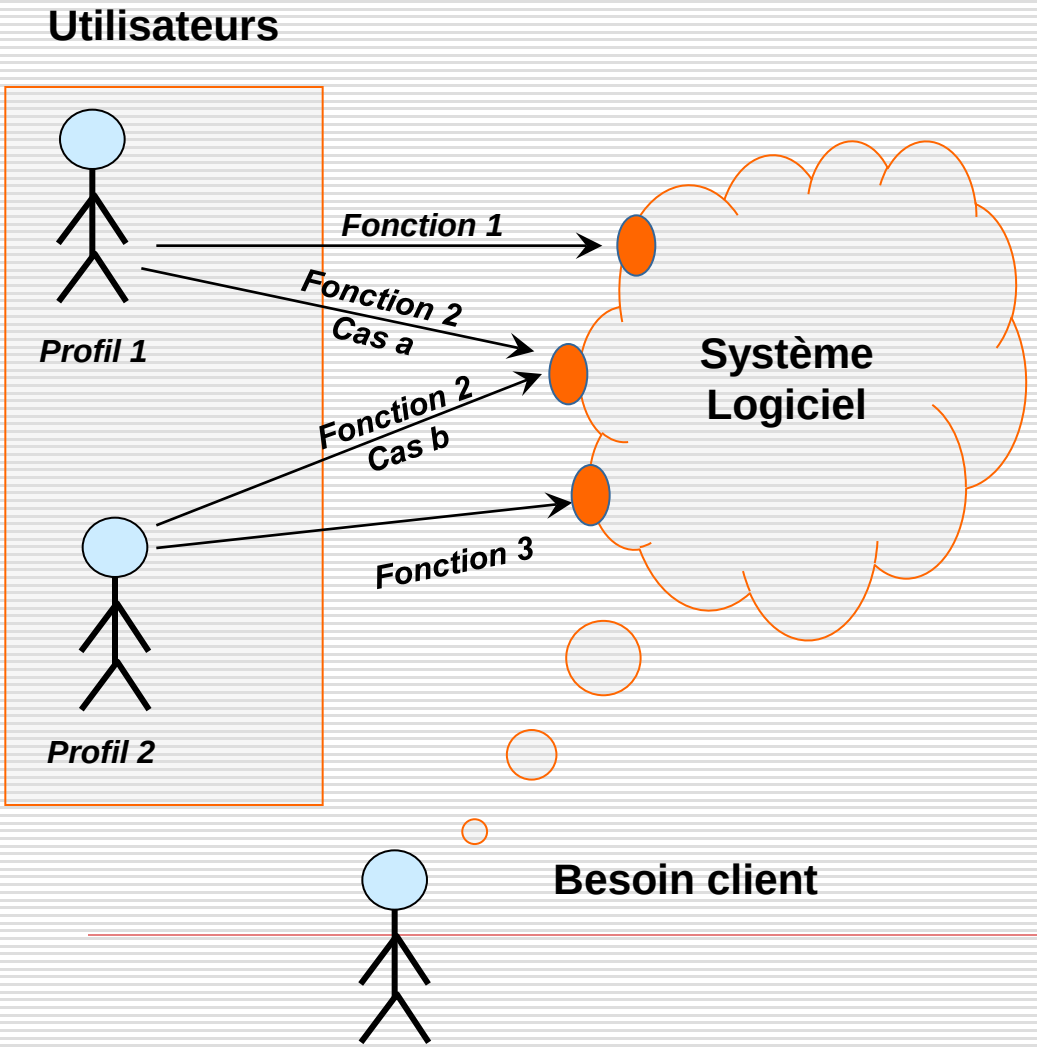
Scénario

Les cas d'utilisation (use case)

- Permet d'exprimer une analyse fonctionnelle du système, cad quels sont les besoins que doit remplir le logiciel et les liens éventuels entre ces besoins.
- Définissent donc ce que doit être capable de réaliser le logiciel face aux demandes des utilisateurs. *Build the right system*
- Doit être simple pour faciliter le dialogue…
- … sans se poser la question (pour l'instant) de la manière de le faire.

- => Correspond à la phase d'ingénierie des besoins : identifier le périmètre de l'étude et les besoins auxquels on doit répondre.

Les cas d'utilisation (use case)



Objectif :

Permettre d'exprimer une analyse fonctionnelle du système, cad quels sont les besoins que doit remplir le logiciel et les liens éventuels entre ces besoins.

On identifie les points d'interaction entre les utilisateurs et le système ie les fonctions qui seront sollicitées par les utilisateurs

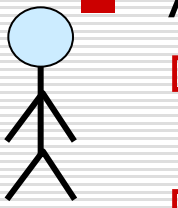
Les cas d'utilisation (use case)

- Définissent donc ce que doit être capable de réaliser le logiciel face aux demandes des utilisateurs. *Build the right system*
- Doit être simple pour faciliter le dialogue…
- … sans se poser la question (pour l'instant) de la manière de le faire.

- => Correspond à la phase d'ingénierie des besoins : identifier le périmètre de l'étude et les besoins auxquels on doit répondre.

Les cas d'utilisation ***Concepts***

- Un « use case » s'appuie sur les concepts de :
  - Acteur :



<<actor>>
WebService

    - entité qui agit sur le système informatique (produit des stimuli/contrôle)
    - Représente aussi bien des acteurs « humains » que des entités matérielles et logicielles interagissant avec le système.
    - Spécialisation/hiérarchisation d'acteurs (stéréotype et héritage)

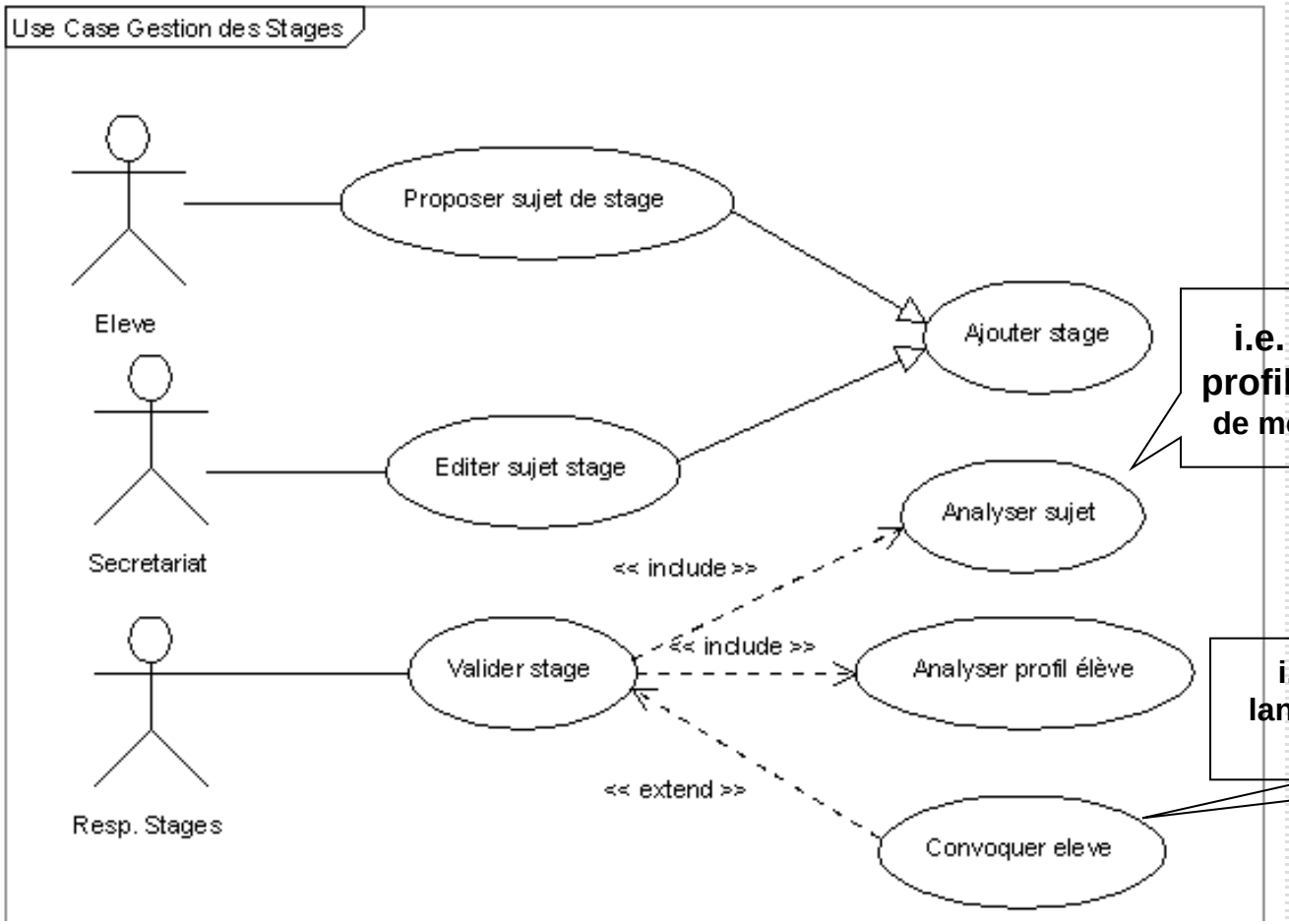
Cas d'utilisation / Use case :

- Ensemble d'actions réalisées par le système
- Elles peuvent être déclinées/spécialisées elles-mêmes en sous use case,
- Regroupé en paquetage (package)

Les cas d'utilisation ***Concepts***

- Un « use case » s'appuie sur les concepts de :
  - Relations entre acteurs :
    - **Héritage** : permet de définir des comportement généraux valable pour plusieurs acteurs et préciser au niveaux des acteurs « enfants » leur spécificité.
  - Relations entre cas d'utilisation :
    - **Généralisation** (héritage) : même principe que pour les acteurs
    - **Inclusion**: traduit la (dé)composition de use case (UC). L'inclusion implique alors que la réalisation d'un UC requiert celle de tous les UC qu'il inclut.
    - **Extension** : comportement non obligatoire (une condition peut être spécifiée) permettant de compléter la description d'un UC.

Les cas d'utilisation ***Exemple***

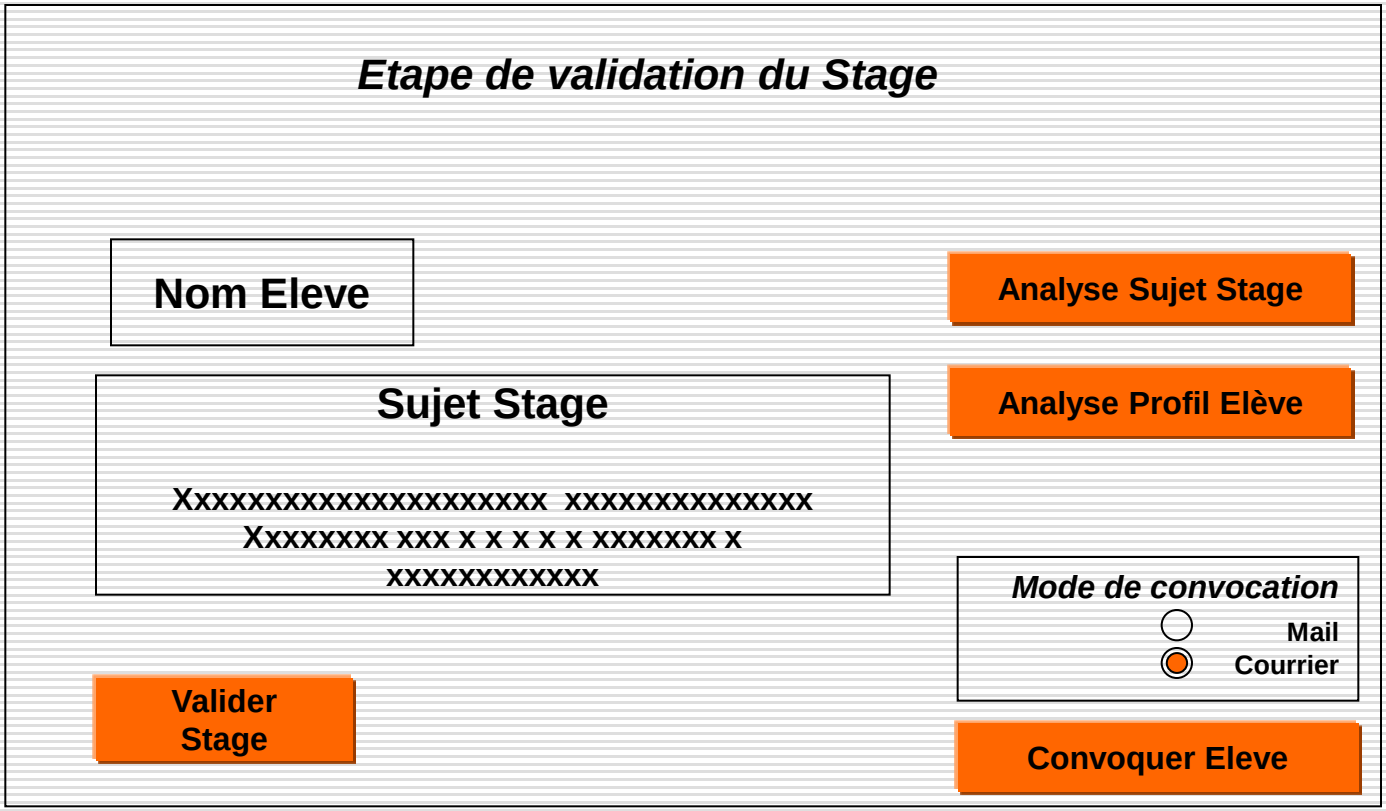


i.e. lancement d'une analyse du profil (qui lancera par ex. une extraction de mots clefs puis une classification…)

i.e. convocation électronique ou lancement impression des courriers « papiers » de convocation

Ces C.U. correspondent à des fonctions invocable par les utilisateurs => implique IHM et NON PAS (en tout cas ici) des fonctions non informatisées ou internes à l'application

Illustration traduction IHM



En fonction du profil d'utilisateur, l'interface pourra varier (droits)

Les pièges des Uses Cases

- Problèmes du niveau de détail :
  - Mélanger les aspects fonctionnels et logiciels
  - Vouloir tout décrire et au final confondre fonction système vers l'utilisateur & fonction interne (cf. ci-après).

- Dériver progressivement vers une description fonctionnelle type SADT :
  - Ex. Cas d'utilisation « Optimiser planning » qui propose un sous cas : « sélectionner goulot d'étranglement ».

2. Vue Dynamique

Diagramme de séquence

Diagramme d’Activité

Les diagrammes de séquence

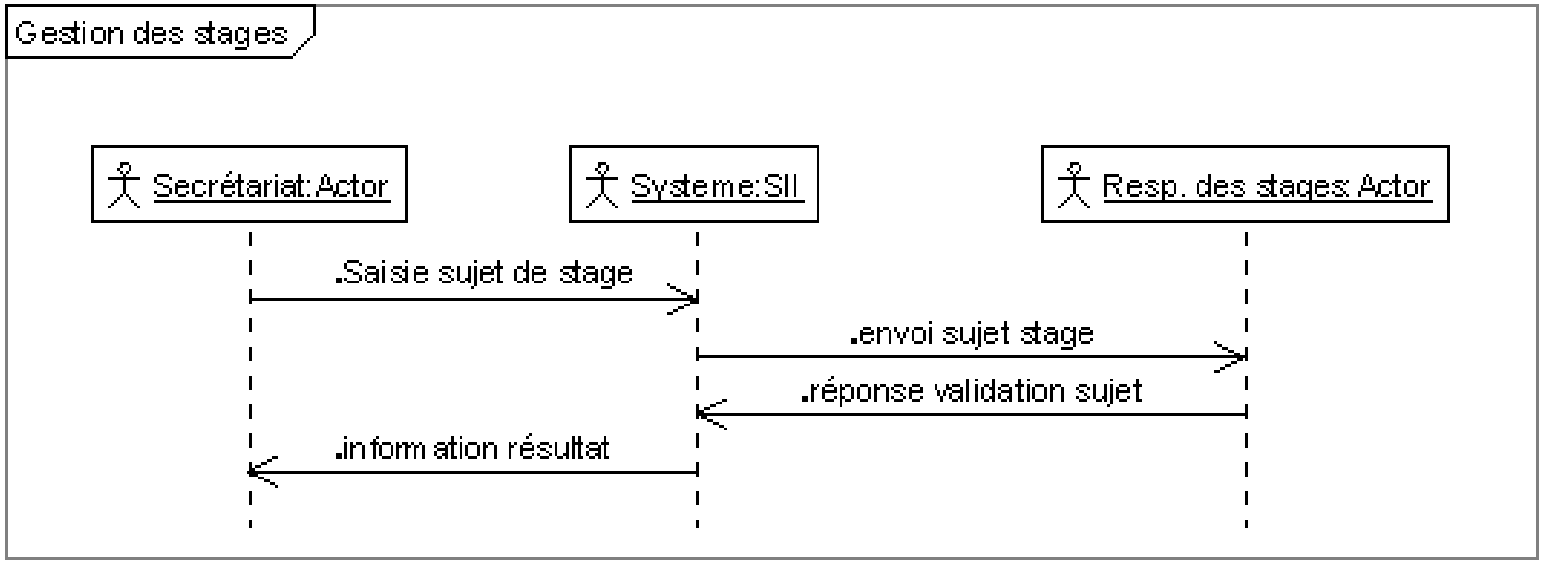
- A la base permet de représenter les échanges de message entre objets (rappel message=appel de méthode)
- Peut également aider à spécifier de manière plus détaillée le fonctionnement du système tel qu'établi avec les cas d'utilisations (CU): ie notion de scénario.
  - Les acteurs peuvent alors être les acteurs du CU et le Système (ou des composants du Système, comme l'IHM & la BD).
  - Les messages traduisent alors les interactions entre ces acteurs.
  - Une ligne verticale dite de vie traduit la chronologie des envois de message

Les diagrammes de séquence

- Type de messages
  - **Synchrone** : le message est bloquant. L'émetteur du message ne reprend ses activités qu'après avoir reçu la réponse au message (ex. appel de fonction)
  - **Asynchrone** : le message n'est pas bloquant, il active une fonction puis retourne à son activité (ex. envoi paquet UDP).
  - **Création** : le message se traduit par une demande d'instanciation d'objet.
  - **Destruction** : le message détruit l'instance (en C++ revient à faire appel au destructeur.
- Durée du message :
  - **Instantanée** : flèche horizontale
  - Avec **un temps de transport** (ex. délais réseau) : flèche oblique

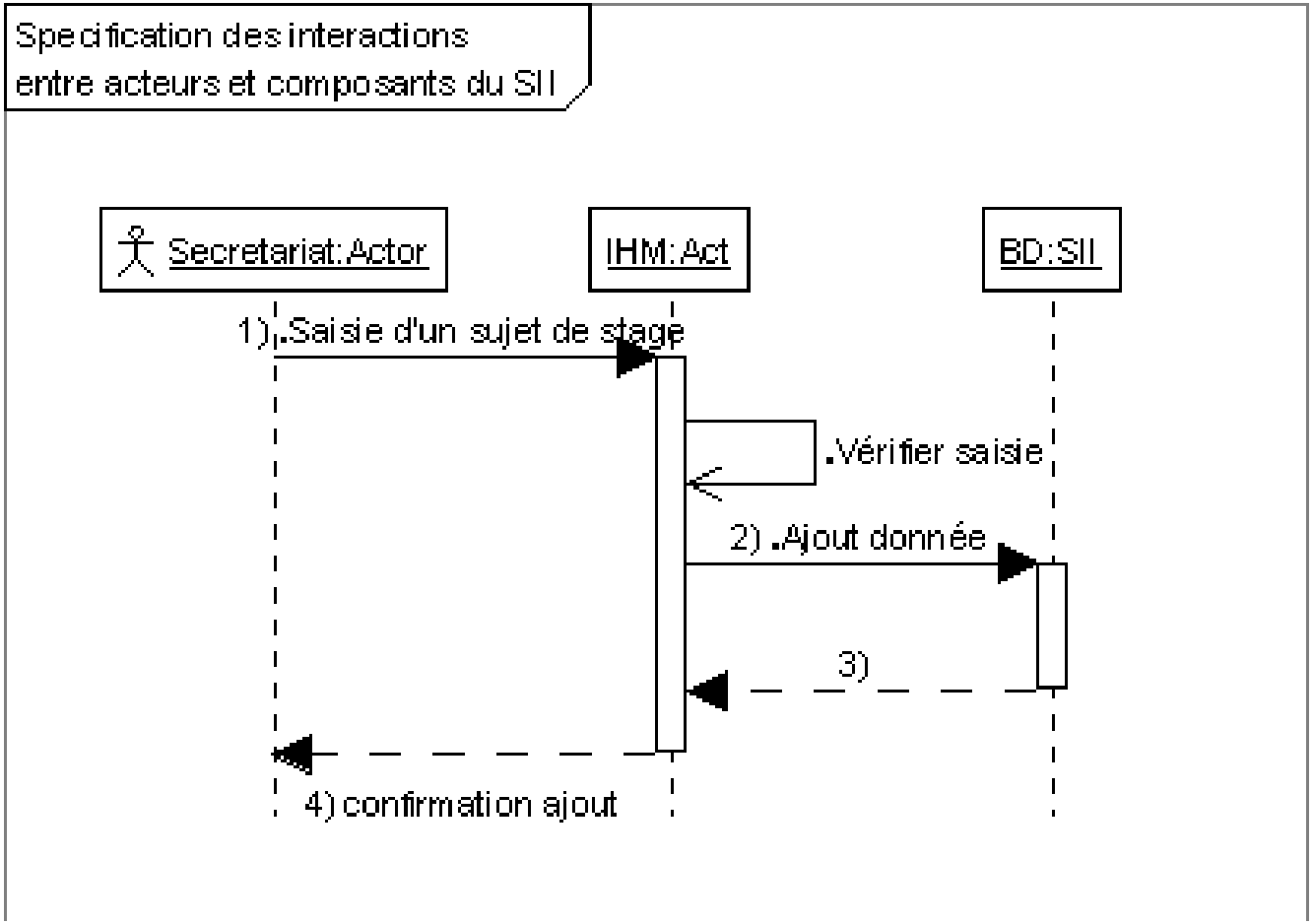
Exemple 1

- Peut servir à représenter l'équivalent d'un diagramme de flux (alors on ne retient que les acteurs du domaine) auquel on aurait rajouté la dimension chronologique des envois de messages.



Exemple 2

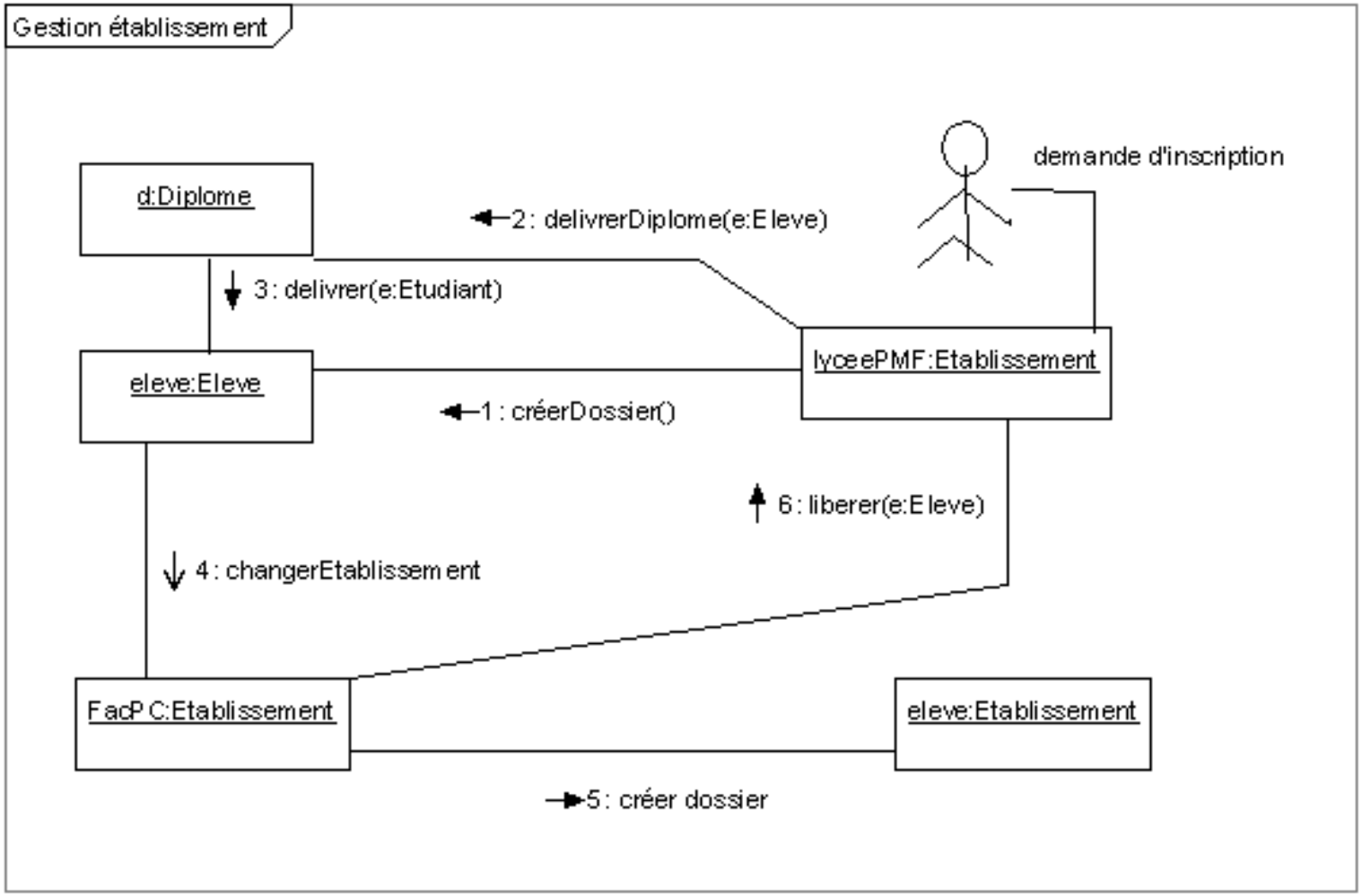
- Analyse du comportement interne du système



Diagrammes de collaboration

- Comparable aux diagramme de séquence en ce qu'il décrivent les interactions entre les classes…
- … mais il s'attache d'avantage à la structure spatiale qu'à l'aspect temporel.
- Permettent de rapprocher le lien entre architecture logicielle et expression des besoins (cas d'utilisation)
- Peuvent s'appliquer :
  - Aux classes : spécification de l'architecture logicielle de l'application
  - Aux instances : conception et description détaillée des interactions en vue de leur implémentation.

Exemple de Diagrammes de collaboration Objet



3. Diagrammes Statiques UML

Diagrammes de classes

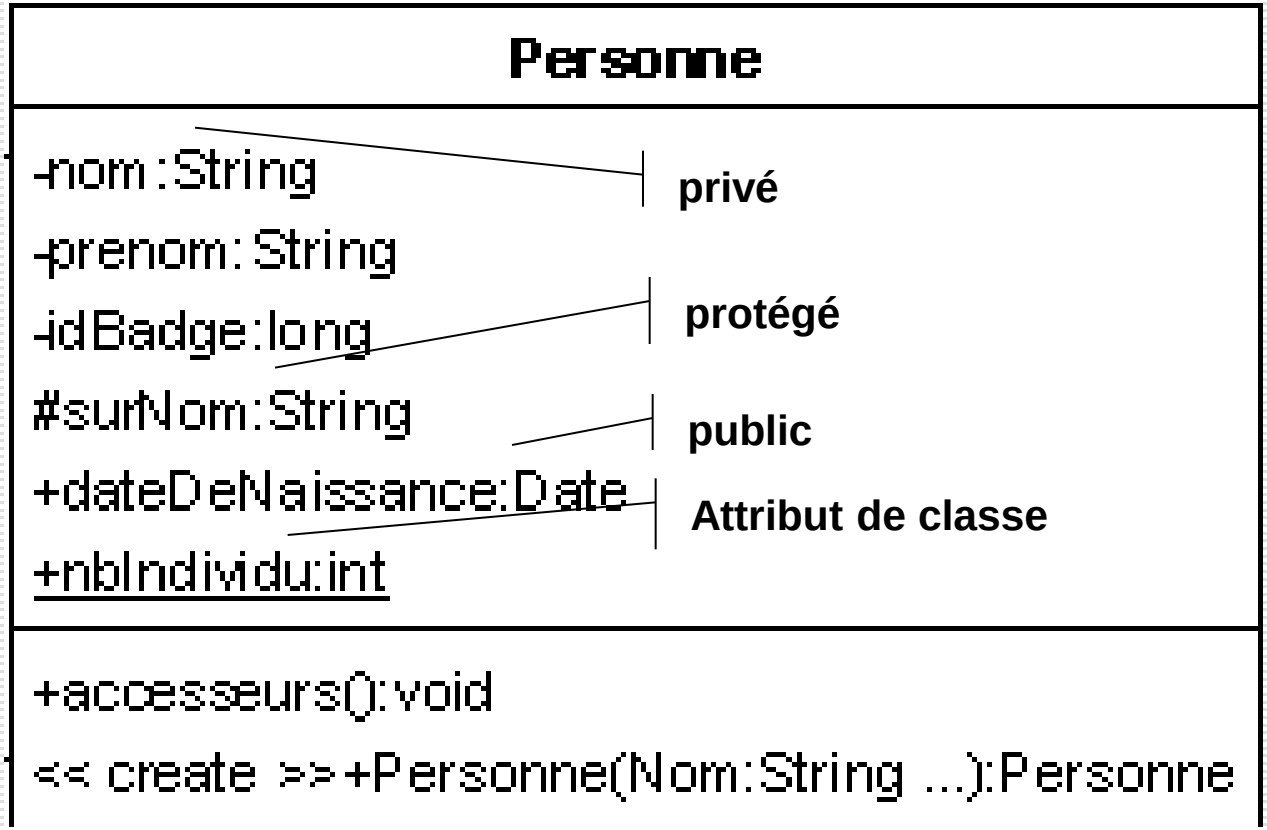
Diagramme de classe

- Classique en OO.
- Décrire la structure du système en terme de classes et de relations entre les classes (héritage, utilisation, composition ...).
- Peut être utilisé pour faire du prototypage

Diagramme de classe ***Concepts***

- Classes
  - Nom
  - Attribut
  - Méthode
- Liens entre classes
  - Héritage
  - Association
    - Simple
    - Héritage
    - Composition/agrégation
  - Relation
    - Utilisation

Exemple de classe



Association

- Signification :
  - traduit des relations structurelles entre 2 classes, ie qui se retrouveront au niveau de leur implémentation (ex. ajout d'un attribut)
  - Il est possible de contraindre l'existence d'une association. Marquée entre { } elle peut s'appuyer sur un langage de contrainte (OCL).
- Symbole :
  - **ligne** (lecture par défaut de gauche a droite), avec éventuellement des flèches pour indiquer des sens de navigation.

Agrégation

- Signification :
  - une occurrence d'une classe peut contrôler une ou plusieurs occurrences d'une autre classe.
- Symbole :
  - **losange** « creux » du côté de la classe propriétaire. (poignée de la laisse)
- La notion d'agrégation signifie que le lien peut être temporaire et rompue sans autres préjudice pour les différentes classes impliquées. (chacun poursuit sa vie)

Composition

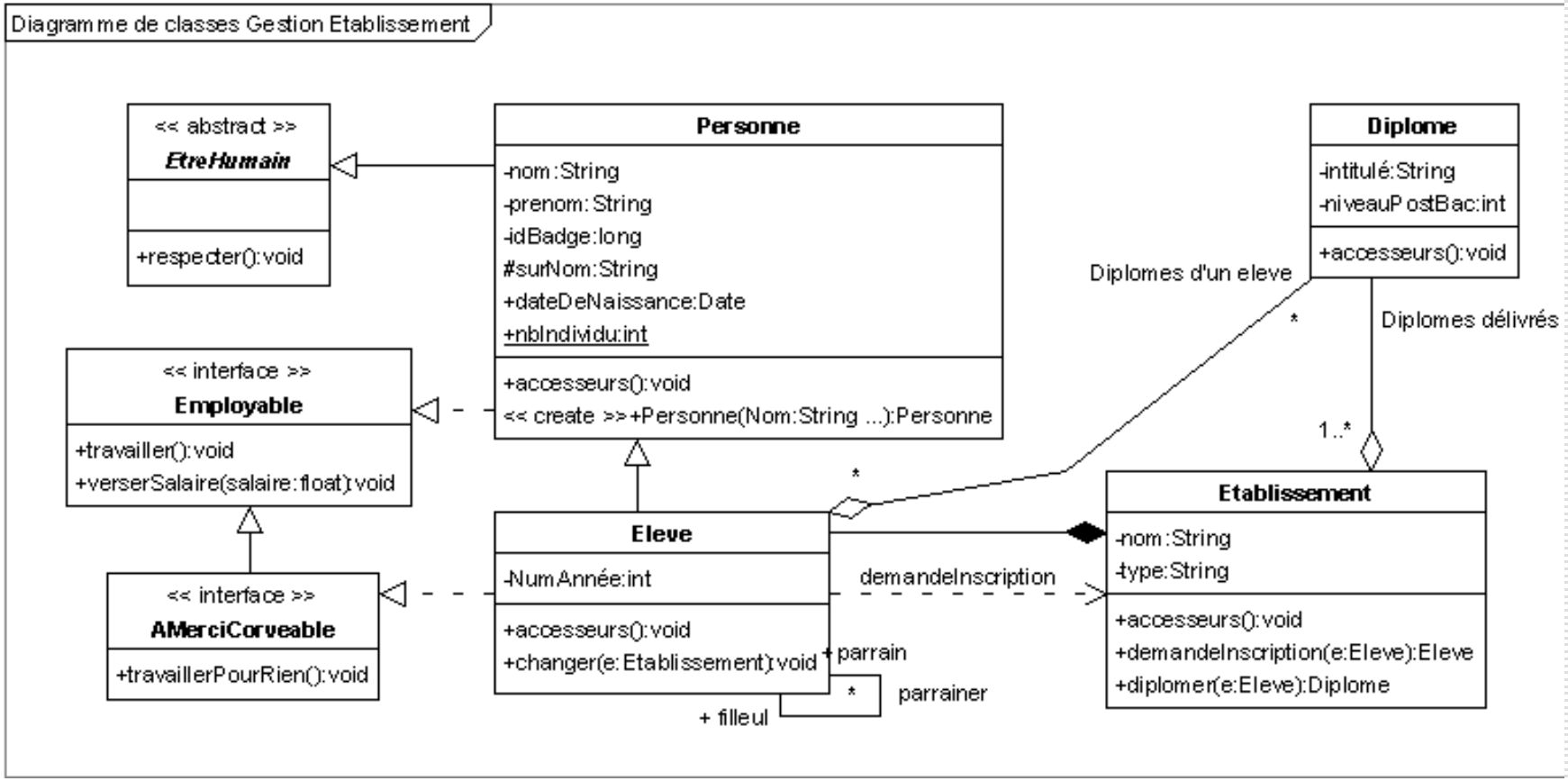
- Signification :
  - une occurrence d'une classe est composée à partir d'une ou plusieurs occurrences d'une autre classe.
- Symbole :
  - **losange plein noir** du côté de la classe qui contient les autres
- La notion de composition est plus forte. Elle implique que les instances des classes qui composent une autre, n'ont d'autres buts que de composer.
- Ainsi, elles ne peuvent survivre à la destruction de la classe d'accueil. Cette dernière les a d'ailleurs sûrement créées.

Relation de dépendance

- Relation entre deux classes qui ne se traduit pas directement dans leur structure (propriétés). Elles sont de 4 ordre : Abstraction (concept), Liaison (entre valeurs), Permission (visibilité, espace de nommage), Utilisation…
- Cas particulier de l'Utilisation : un objet d'une classe peut utiliser celui d'une autre (sans le « posséder »).
  - En modélisation : traduit des relations entre concepts/entité
  - En programmation : fréquemment utilisée pour montrer qu'un objet d'une classe peut appeler les méthodes d'une autre classe.
- Symbole : une flèche pleine.

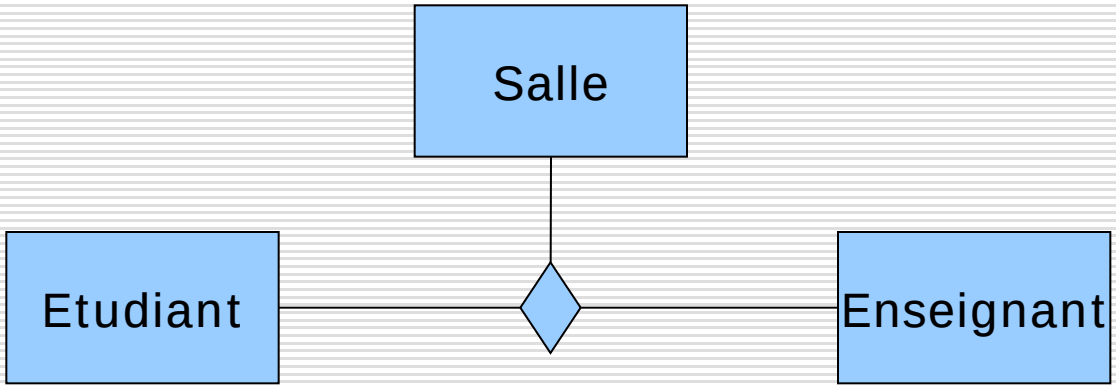
Exemple de diagramme de classe – aspect BD/POO

- Effectifs d'un établissement de formation



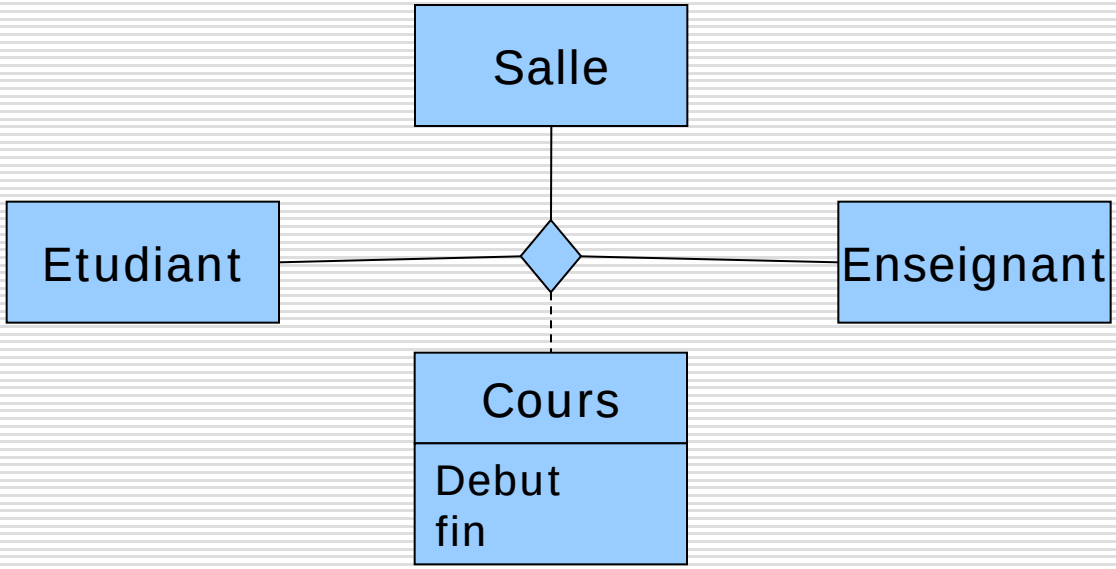
Exemple de diagramme de classe

- Exemple de relation ternaire



Exemple de diagramme de classe

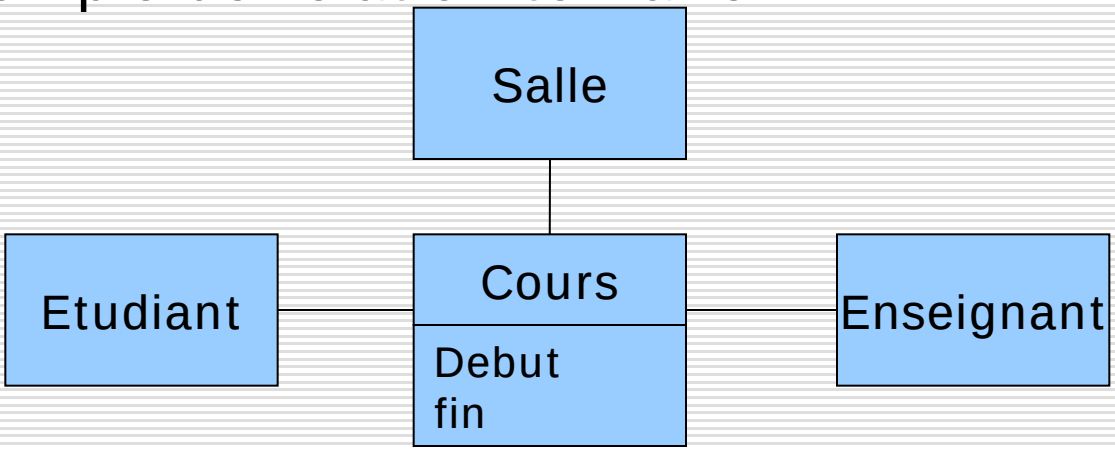
- Exemple de relation ternaire



- On peut caractériser une association par une classe si l'on veut lui apposer des propriétés (ligne pointillée). C'est cependant une classe comme les autres.

Exemple de diagramme de classe

- Exemple de relation ternaire

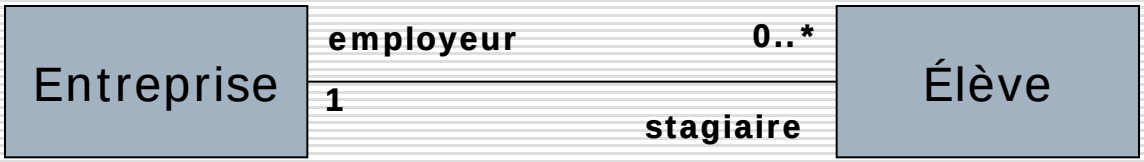


- La relation devient une classe.
- Il faut alors spécifier que toute instanciation de cette classe implique de préciser simultanément les instances des autres classes (revient à définir des clés étrangères…).

Les diagramme de classe

Comparaison Merise

- Spécifier une BD avec UML revient se situe entre le MCD et le MLD relationnel (pas de propriété de relation mais possibilité de le représenter par une classe).
- On retrouve les propriété d'héritage (puisque c'est de la technologie Objet dont s'était inspiré Merise)
- Les cardinalités sont :
  - ouvertes (1, 0..1, n, m..p, *, etc.) … mais cela était le cas au niveau du MOD.
  - Notée de manière inversée par rapport à MERISE



1 entreprise emploie comme stagiaire 0 à n élèves
1 élève effectue son stage dans 1 et 1 seule entreprise

4. Gestion de projet avec UML et retour sur les diagrammes

Vocabulaire projet

Autres diagrammes UML
(diagramme d'état-transition, diagramme d'activité, retour sur les classes ...)

Conception d'applications avec UML

- Gérer la complexité en définissant des couches/niveaux d'abstraction :
  - Couche métier : modélisation de l'activité en évacuant les aspect informatique.
    - Approche classique externe/interne, statique/dynamique
    - correspond aux couches conceptuelles/organisationnelles de MERISE (pas vraiment présent dans Windev).
  - Couche informatique : partie de la couche métier à automatiser. Traduit de manière concrète l'activité des acteurs de l'organisation puisqu'il s'agit de concevoir les applications informatiques supportant leur travail « quotidien ».
    - couche logique/physique de MERISE.

Modélisation du métier

- Correspond à la notion de modélisation de l'organisation dans MERISE. C'est-à-dire modéliser la réalité pour l'automatiser ... et non pas considérer comme réel ce que veut/sait implémenter l'informaticien...
- Ce qui implique de suivre les étapes suivantes:
  - délimiter le champs de l'étude : périmètre, acteurs sur le système
  - Étude des processus
  - Structure interne du système

Utilisation des diagrammes

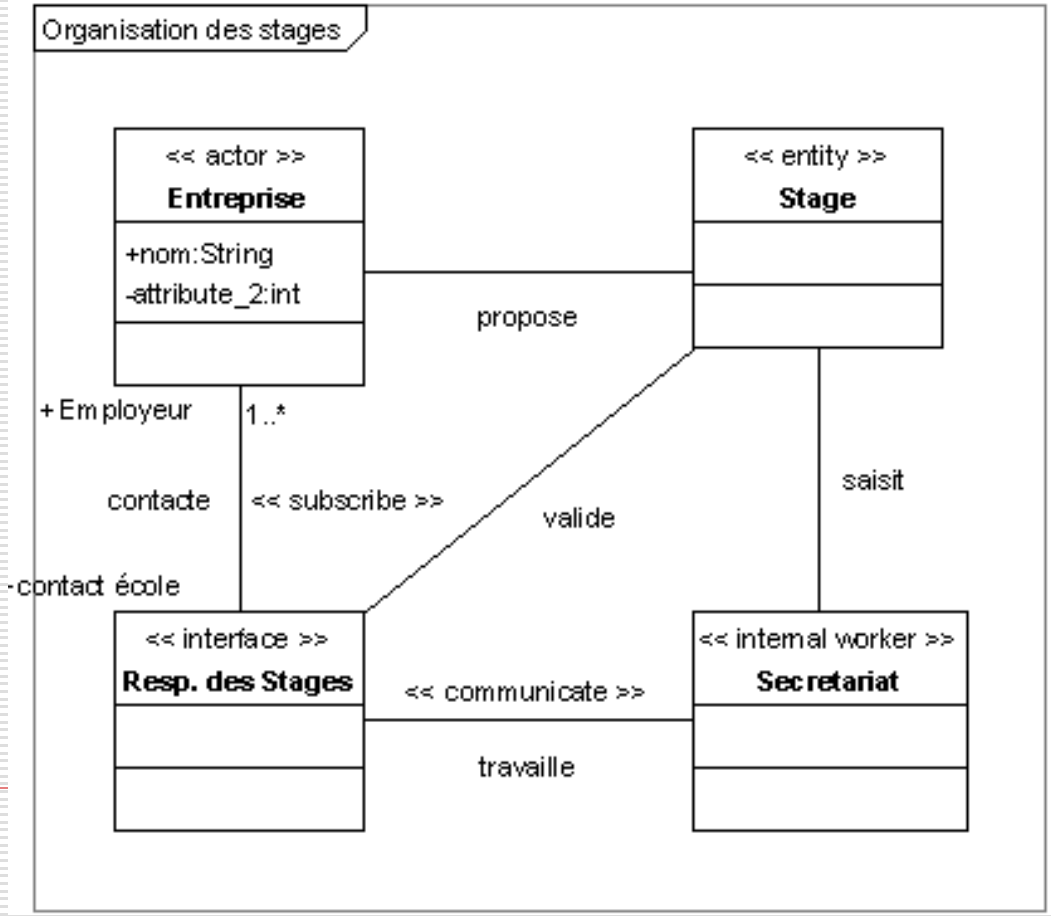
- Il est possible d'utiliser un diagramme de classe pour décrire une organisation.
- Dans ce cas, les classes décrivent les travailleurs de l'organisation ie les acteurs internes au sens Merise.
- Il ne s'agit pas de décrire les interactions ou échanges de message entre les acteurs mais les liens (hiérarchique, dépendance) entre les acteurs.

Retour sur les diagrammes de classe

- Afin d'affiner la modélisation de l'organisation, il est possible de caractériser les classes à l'aide de stéréotype (symbole <<xxx>>) ce qui revient à définir des familles de classes.
- UML fournis des stéréotypes prédéfinis (eg. <<Utilitaire>>) dont les suivants sont plus centrés sur la modélisation métier :
  - **Interface Worker** : acteur **de** l'organisation visible par l'extérieur de l'organisation
  - **Internal Worker** : acteur **de** l'organisation non visible par l'extérieur de l'organisation (petites mains).
  - **Entity**: objet manipulés par les acteurs (ayant de grande chance d'être transposées dans le modèle logique).

Retour sur les diagrammes de classe

- Exemple : si les stages sont envoyés à l'école mais les entreprises ne connaissent que le responsable des stages



Notion de processus métier

- Terme incontournable ! Oblige en quelque sorte l'informaticien à intégrer le point de vue du client …
- une approche peut être 1 cas d'utilisation = 1 processus métier.
  - Ex : gérer stages
    - **Acteurs** : élève, entreprise, responsable des stages, secrétariat
    - **Cas d'utilisation** : celui-ci généralise celui présenté au début de chapitre
- 3 types de processus :
  - Processus métier : ce qui est visible de l'extérieur (du champ de l'étude), ie c'est ce que « vends l'entreprise »
  - Processus support : processus internes à l'entreprise (donc non visible) mais qui assurent l'exécution des processus métiers.
  - Processus de gestion : interne également à l'entreprise, permettent de gérer les processus (ex. processus de processus : pilotage).

Description d'un processus métier

- Textuelle : rôle, intervenants et objectif du processus. Implique d'établir le vocabulaire métier afin d'éviter tout risque de confusion/ambiguïté.
- Formelle :
  - Diagramme d'interaction (diagramme séquence ou de collaboration): établissant les interactions entre les acteurs/objets (au sens large et pas au sens informatique).
  - machine à état d'Harel (*statechart* ou diagramme d'activité dans UML).
  -

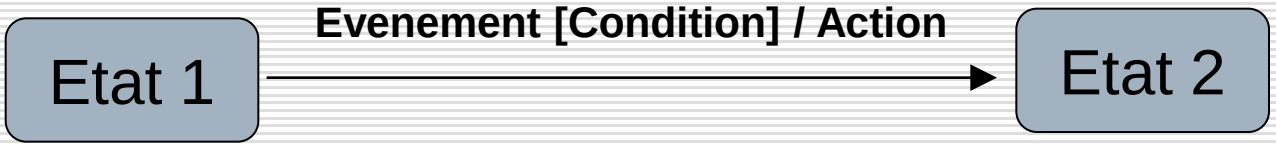
Diagramme de État Transition

Objectif

- Permet de décrire le cycle de vie d'un Objet :
  - Les différents états qu'il peut atteindre
  - Les règles régissant le passage d'un état à un autre
- Au niveau métier permet de décrire les règles de gestion assurant comment sont utilisés les objets du monde (ex. facture).
- Au niveau logique :
  - Transposition au niveau des objets concrets (ie futur code)
  - Spécification de règles de comportement des objets « concrets » (ex. objet graphique).

Diagramme de État Transition

Concepts



- **État :**
  - Objet: caractérisé en principe par l'ensemble des valeurs prises par les attributs de l'objet à un instant t. Mais selon cas peut se réduire à l'étude de certains attributs.
  - Métier: caractérise l'état d'une entité au sens large : état stable, en cours d'exécution, attente d'un évènement…
- **Transition :**
  - traduit le passage possible d'un état à un autre avec éventuellement des conditions sur l'événement à l'origine de l'activation de la transition (condition de garde).

Diagramme de État Transition

Concepts

- **Type d'État :**
  - **État initial** : traduit l'initialisation du diagramme d'état (1 seul par diagramme).
  - **État final** : caractérise la fin du diagramme (0..n).
  - **État simple** : caractérise l'état d'un objet
  - **État composite**: permet de gérer l'abstraction en définissant un « super » état dans lequel une séquence d'état et de transition peuvent décrire plus finement ce qui se passe dans cet état (ex. l'état Inscription peut être décomposé en état saisi, état frais inscription calculés, frais payés, inscrit).

Diagramme de État Transition

Concepts

- **Type d'évènement pouvant déclencher une transition :**
  - **Réception d'un signal** : envoi d'un message asynchrone par un autre objet ou par un acteur (point de vue métier)
  - **Appel de méthode** (*call event)* sur l'objet modélisé (synchrone ou asynchrone). (point de vue objet)
  - **Écoulement d'une durée** (*time event)* relative (10 sec) ou absolue (date précise). Utilisation du mot clé after
  - **Changement de condition** (*change event*) : décrite par une expression booléenne sur des variables (état d'une variable, d'un objet, d'une valeur globale) mot clé when

Diagramme de État Transition

Exemple

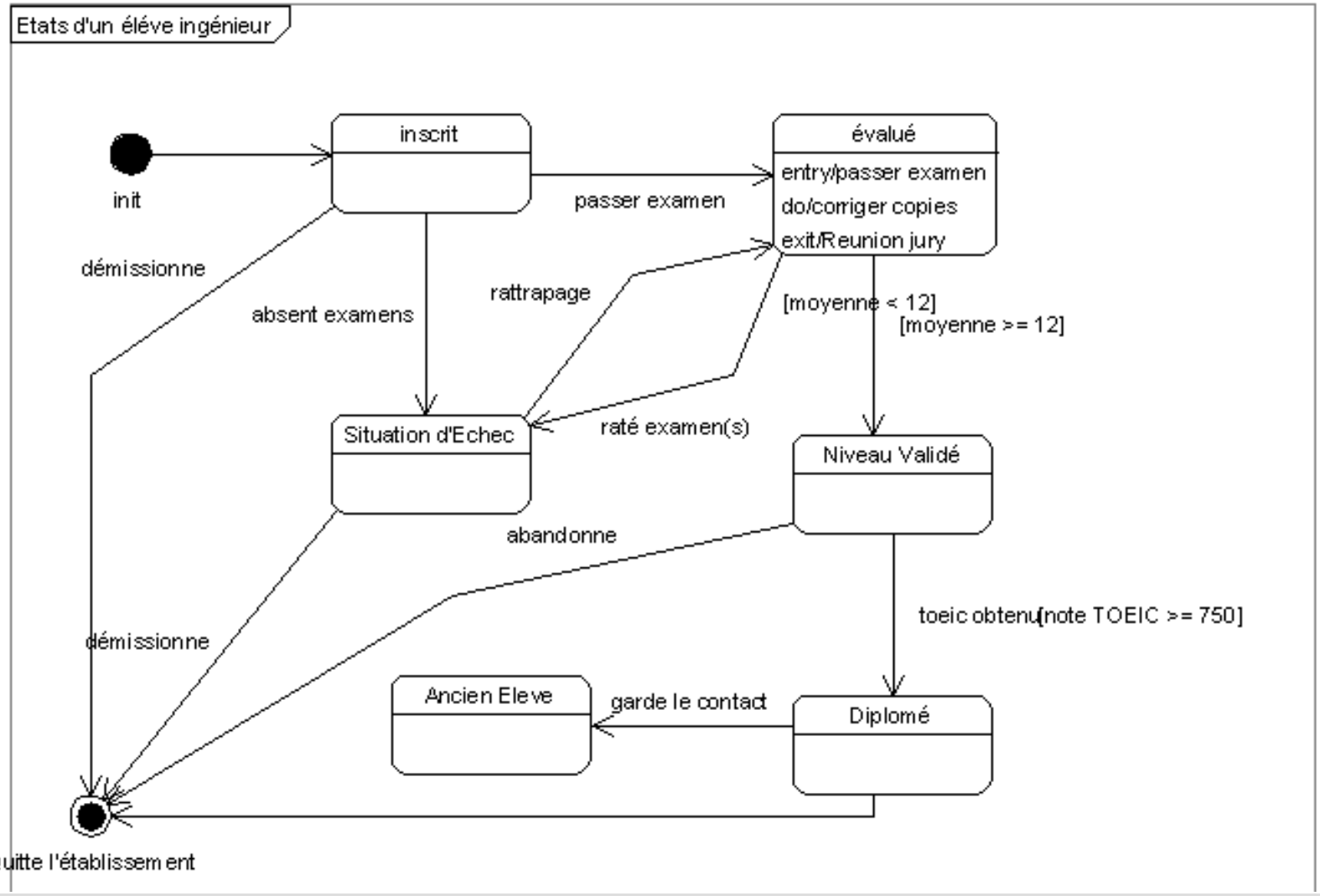


Diagramme de État Transition

Concepts

- **Diagramme actif**: possibilité de spécifier des actions…
  - **Action : opération associée à un état non interruptible**
  - **Activité : idem mais interruptible.**
- **Ces opérations peuvent être exécutées :**
  - **Dés l'entrée dans l'état**
  - **A la sortie de l'état**
  - **Pendant l'activation de l'état (interruptible)**
- **Les transitions peuvent également réaliser des actions**

Inscription
Entrée/vérifier identité Faire/inscrire Sortie/clôturer session

Diagramme de État Transition

Aspect non développés ici

- **Diagrammes hiérarchiques pour gérer la complexité (état composite)**
- **Héritage sur les transition**
- **Représentation de processus parallèles (avec états concurrent, synchronisation)**
- **…**

Diagramme d'activité

Objectifs

- Spécialisation (au sens héritage) des diagrammes d'état permettant à la manière d'un organigramme de décrire des séquences d'actions et éventuellement les responsables de leur exécution.
- Comparaison MERISE : entre le MCT et le MOT (ie MOT non détaillé).
- Peut être utilisé pour décrire/spécifier un cas d'utilisation. Niveau d'abstraction plus détaillé.
- Sont gérés :
  - les événements externes (comme les diagramme d'états-transitions)
  - les événements internes, ie résultant des actions des « objets » interne du système.
  - La composabilité et la récursivité (1 diagramme d'activité peut être utilisé pour décrire un autre)

Diagramme d'activité

Concept

- **État action** (action state) : état auquel on associe une action (traitement) interne et au moins 1 transition sortante (indiquant la fin de l'action).
- **(Nœud de) décision :** si plusieurs issues de l'action sont possibles, il devient nécessaire de spécifier les conditions d'activation (ou conditions de garde) des transitions afin de déterminer laquelle effectivement déclencher.
  =>conditions mutuellement exclusive !
- **Partition :** « couloirs » permettant de caractériser les objets/acteurs en charge des différentes actions. ~ notion de poste de travail du MOT

Diagramme d'activité

- Objet : les objets (ie stéréotype entity) peuvent être indiqué afin de préciser les conséquences des actions sur l'état du système.

- Condition de garde : sont décrite sous la forme d'expression booléenne. Il est possible d'améliorer la rigueur de la modélisation en utilisant le langage de contrainte OCL (Object Constraint Language)

Diagramme d'activité

Exemple simple

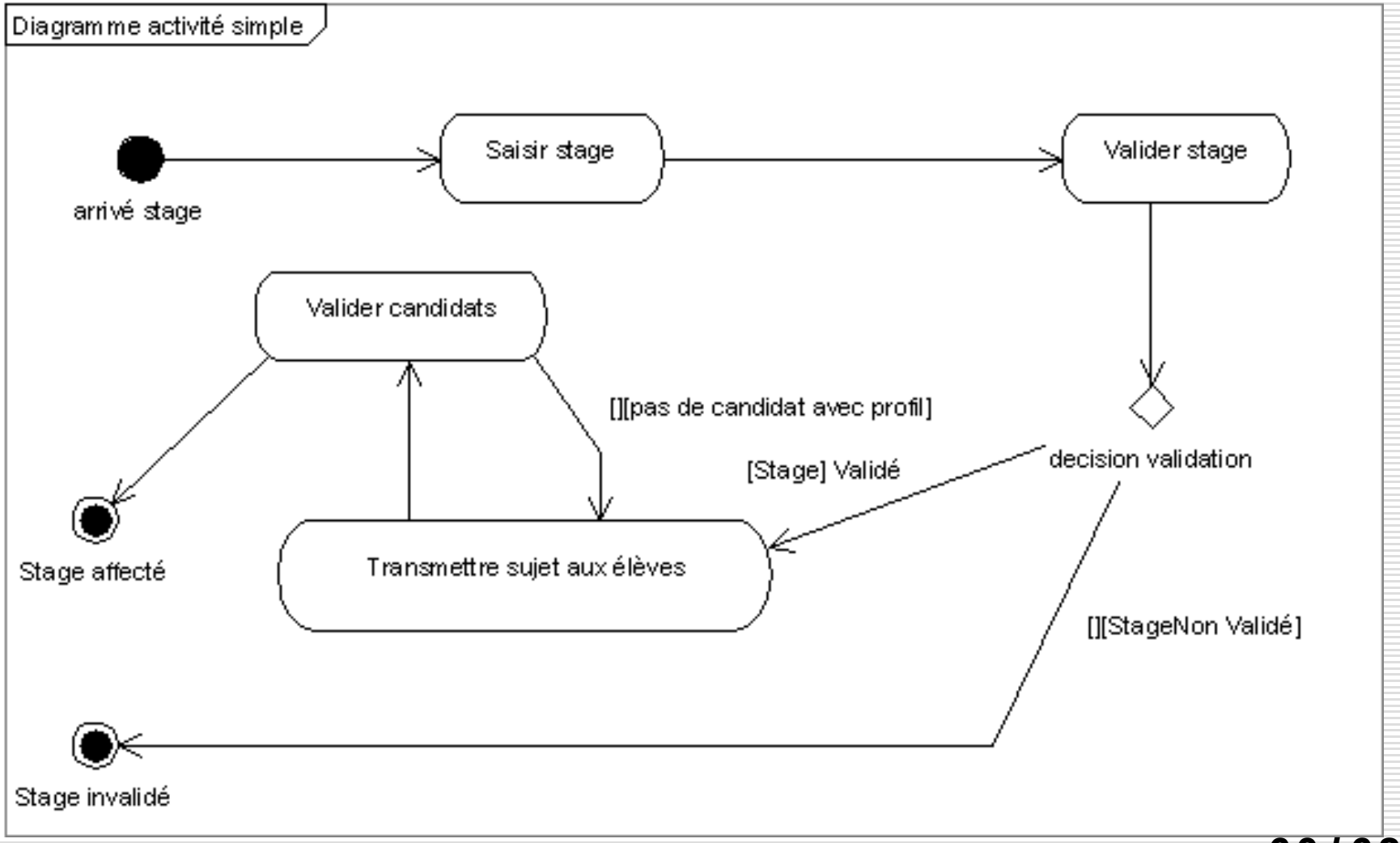
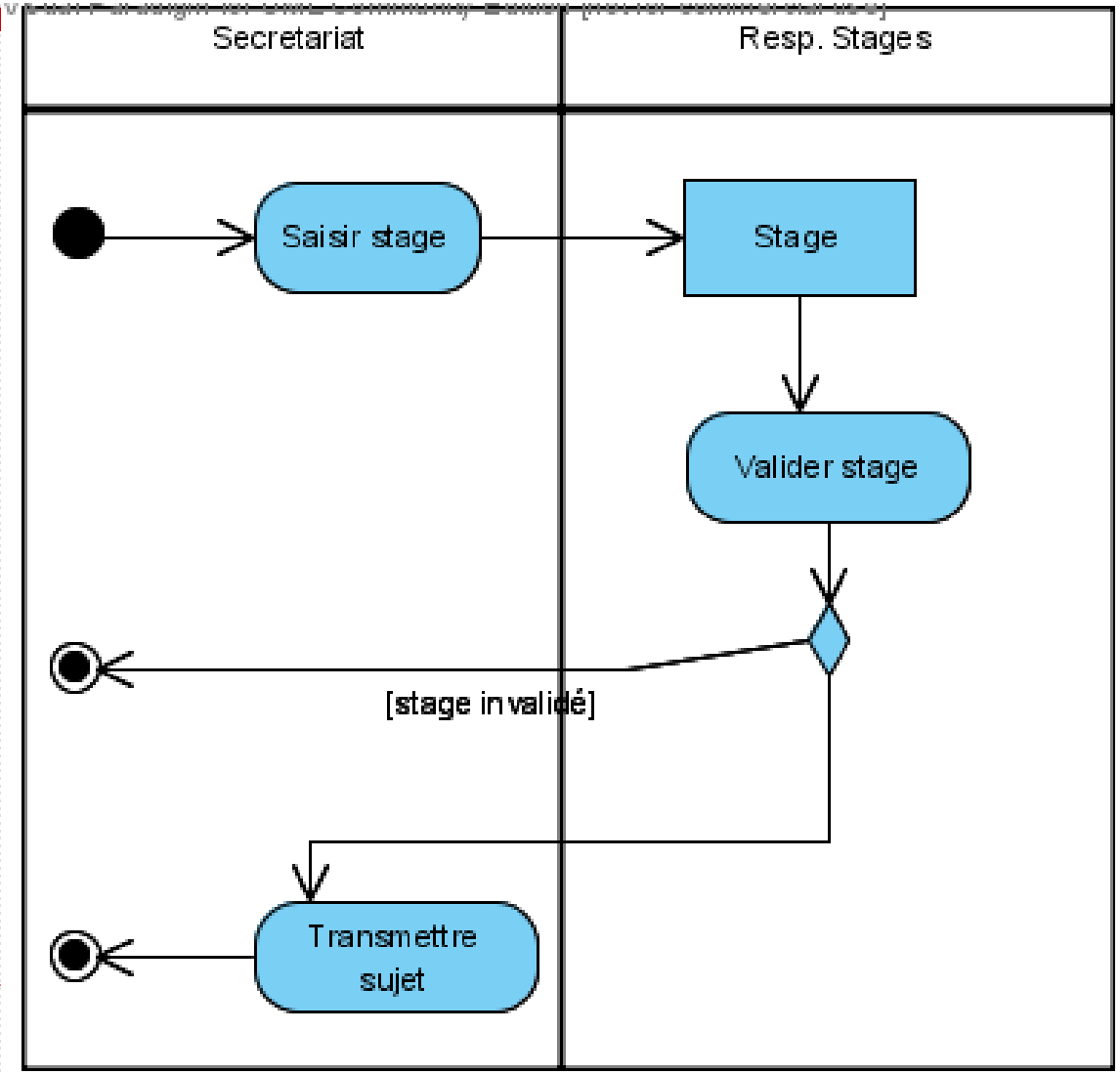
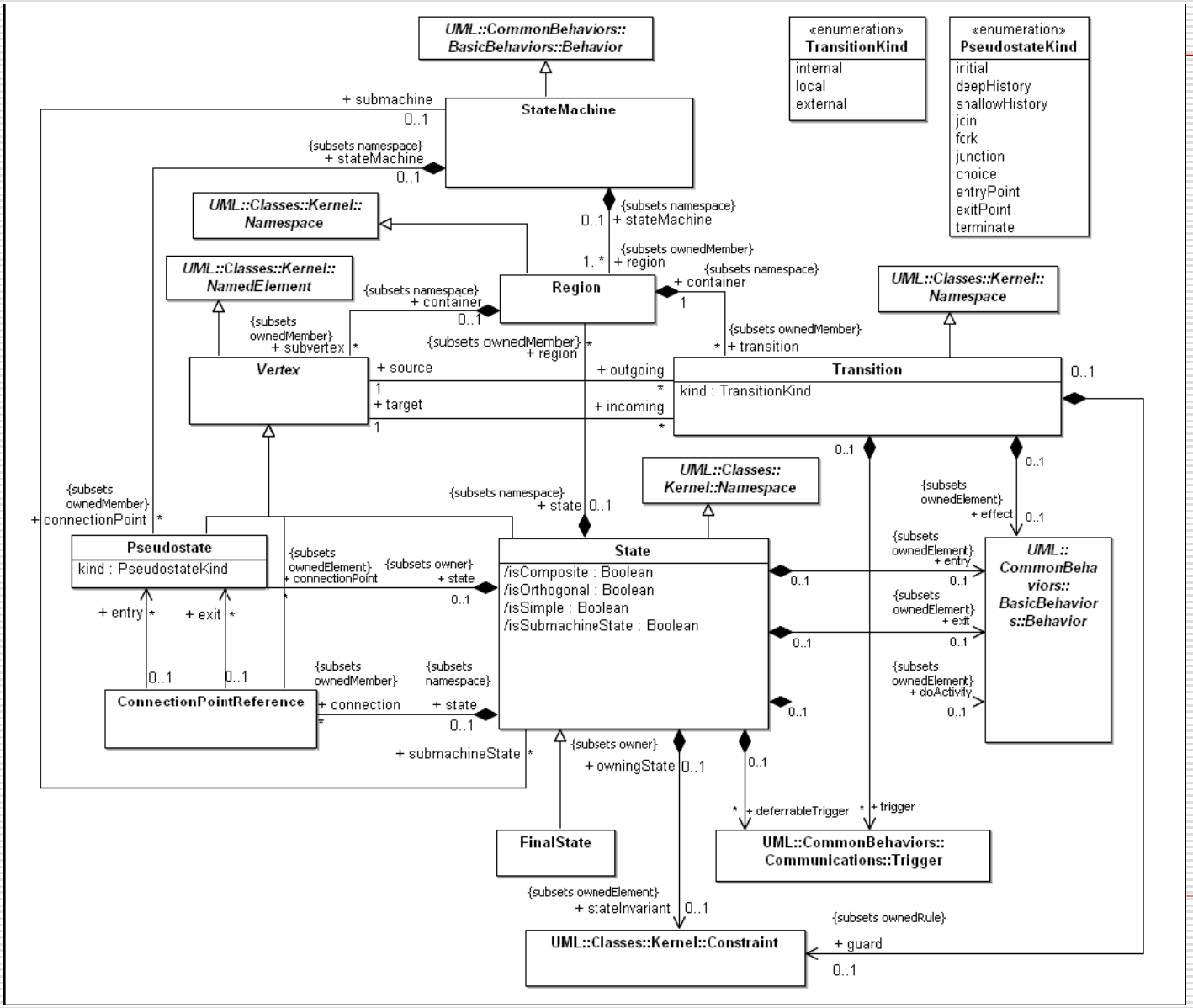


Diagramme d'activité

Exemple avec Couloir



Annexe – Réflexion sur les métamodeles…

Ex. Machine à état